Ledesma Ayala Angel Yeremi

API Collections:

El API Collections en Java es una parte fundamental de la biblioteca estándar (java.util) que proporciona un conjunto de clases e interfaces para trabajar con colecciones de datos. Estas colecciones incluyen listas, conjuntos, mapas y otras estructuras de datos que son ampliamente utilizadas en la programación. Aquí hay una breve introducción a sus características generales y cómo importar la biblioteca:

Características Generales del API Collections en Java:

1. Interfaz y Clases: El API Collections define una serie de interfaces, como List, Set, y Map, que representan tipos de colecciones genéricas. Además, proporciona implementaciones concretas de estas interfaces, como ArrayList, HashSet, y HashMap.
2. Tipos de Datos Genéricos: Las colecciones en Java pueden ser parametrizadas con tipos de datos genéricos, lo que permite trabajar con cualquier tipo de objeto de manera segura y tipada.
3. Operaciones Comunes: El API Collections ofrece operaciones comunes para agregar, eliminar, buscar y manipular elementos en colecciones. Esto facilita la gestión de datos de manera efectiva.
4. Iteración: Proporciona métodos para iterar de manera eficiente a través de los elementos de una colección utilizando bucles o iteradores.
5. Ordenación y Orden: Algunas clases de colecciones, como TreeSet y TreeMap, mantienen un orden específico de sus elementos. Otras, como HashSet y HashMap, no garantizan un orden específico.
6. Colecciones Sincronizadas: Se ofrecen versiones sincronizadas de las colecciones para garantizar la seguridad en entornos con múltiples hilos de ejecución.

Importación del API Collections en Java:

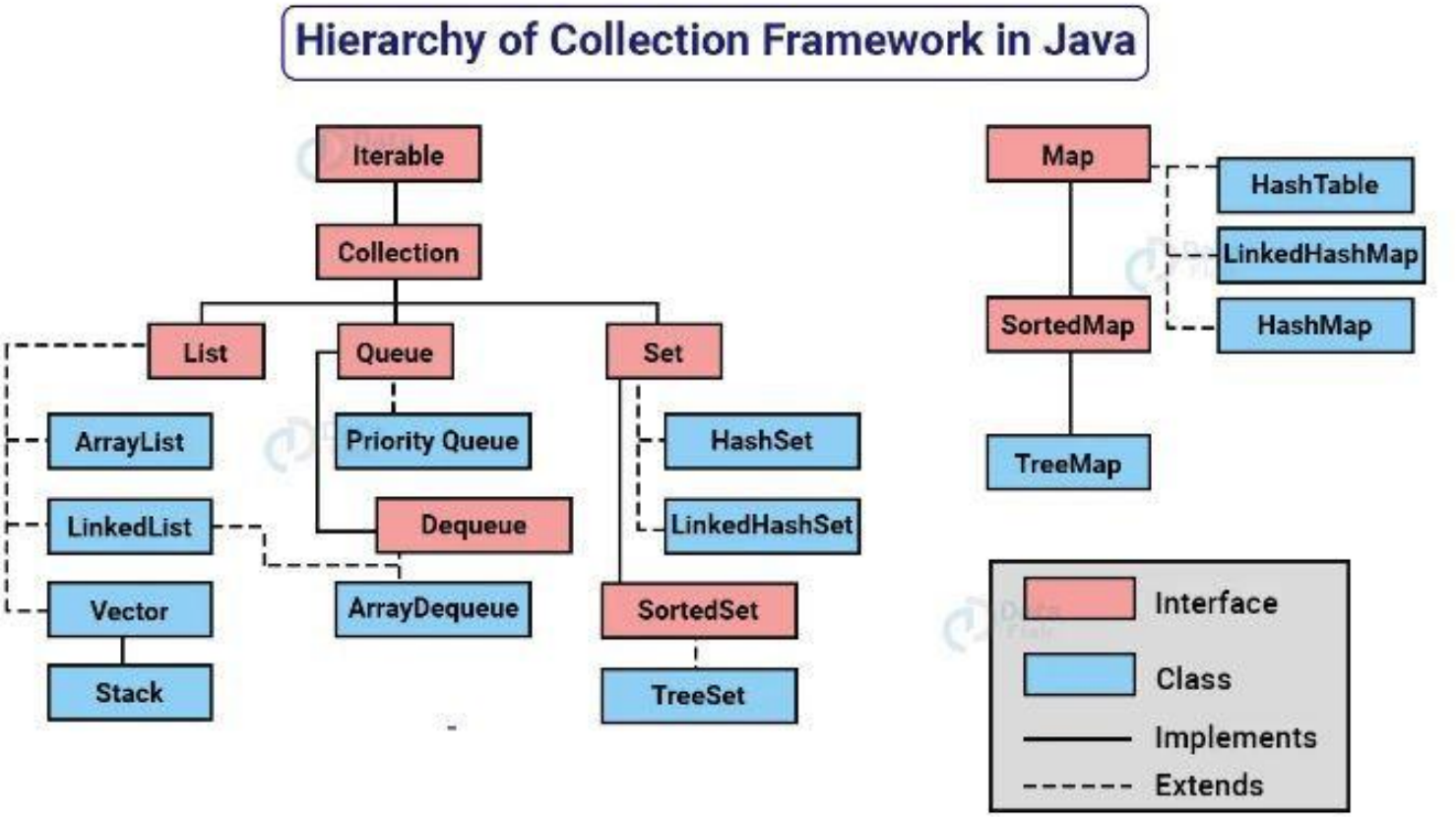
Para utilizar el API Collections en Java, se debe importar la biblioteca java.util. Esto se hace en el programa mediante la declaración siguiente:

```
import java.util.*;
```

Con esta declaración, se tendrán acceso a todas las clases e interfaces relacionadas con las colecciones en Java. A partir de ese punto, se pueden utilizar las clases y métodos proporcionados por java.util para trabajar con colecciones de datos en Java de manera efectiva.

En resumen, el API Collections en Java proporciona una amplia gama de estructuras de datos y operaciones para gestionar y manipular datos de manera

eficiente. Su uso es esencial en el desarrollo de aplicaciones Java que requieren almacenamiento y manipulación de colecciones de datos.



List:

En Java, List es una interfaz que representa una colección de elementos ordenados que pueden contener duplicados. Es ampliamente utilizada para almacenar y manipular listas de elementos. Aquí se destacan sus características y los métodos más utilizados e importantes para operaciones comunes:

Características Principales de List en Java:

- Orden: Los elementos en una List se mantienen en un orden específico, lo que permite acceder a ellos por su posición en la lista mediante un índice entero.
- Duplicados Permitidos: A diferencia de algunas colecciones como Set, las listas permiten duplicados, lo que significa que puedes tener múltiples elementos idénticos en una lista.

- Tamaño Dinámico: Las listas en Java tienen un tamaño dinámico, lo que facilita agregar o eliminar elementos en cualquier momento sin especificar un tamaño fijo.

Implementaciones de List en Java:

- ArrayList: Esta implementación almacena elementos en un arreglo dinámico, lo que permite un acceso rápido por índice. Es eficiente para lecturas y búsquedas, pero puede ser menos eficiente para inserciones y eliminaciones en el medio de la lista.
- LinkedList: Almacena elementos en nodos enlazados, lo que facilita inserciones y eliminaciones en cualquier parte de la lista. Es eficiente para agregar o quitar elementos al principio o al final de la lista, pero puede ser más lento en acceso aleatorio.
- Vector: Similar a ArrayList, pero sincronizado para su uso en entornos multihilo. Puede ser menos eficiente que ArrayList en entornos de un solo hilo debido a la sincronización.

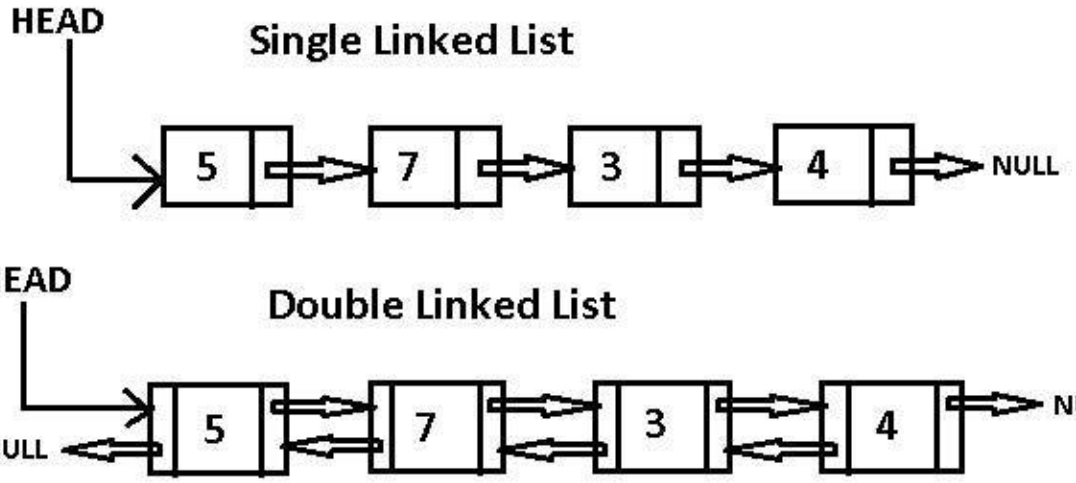
Métodos Importantes en List:

- add(E elemento): Agrega un elemento al final de la lista.
- add(int índice, E elemento): Agrega un elemento en la posición especificada.
- remove(Object objeto): Elimina la primera ocurrencia del objeto dado.
- remove(int índice): Elimina el elemento en la posición especificada.
- get(int índice): Obtiene el elemento en la posición especificada.
- set(int índice, E elemento): Reemplaza el elemento en la posición especificada con el nuevo elemento.
- size(): Devuelve el número de elementos en la lista.
- isEmpty(): Verifica si la lista está vacía.
- contains(Object objeto): Verifica si la lista contiene el objeto dado.
- indexOf(Object objeto): Devuelve el índice de la primera ocurrencia del objeto dado.
- lastIndexOf(Object objeto): Devuelve el índice de la última ocurrencia del objeto dado.

Eficiencia y Uso de List en Comparación con Otros Tipos de Colecciones:

- Acceso por Índice: List es eficiente cuando necesitas acceder a elementos por su índice, siendo tanto ArrayList como LinkedList adecuados para esto.

- Inserciones y Eliminaciones: LinkedList es eficiente para inserciones y eliminaciones en cualquier parte de la lista, mientras que ArrayList es más eficiente para agregar o quitar elementos al final de la lista.
- Memoria: ArrayList suele ser más eficiente en términos de uso de memoria en comparación con LinkedList.
- Iteración: ArrayList tiende a ser más eficiente en la iteración debido a su estructura de arreglo contiguo.
- Escenarios Específicos: La elección entre ArrayList, LinkedList, u otras implementaciones de List dependerá de las necesidades específicas de tu aplicación, considerando acceso, inserciones, eliminaciones, uso de memoria y otros factores de rendimiento.



Set:

En Java, Set es una interfaz que representa una colección de elementos únicos, lo que significa que no permite elementos duplicados. Aquí se destacan sus características y los métodos más utilizados e importantes para operaciones comunes:

Características Principales de Set en Java:

- Elementos Únicos: En un Set, no se pueden tener elementos duplicados. Cada elemento en el conjunto es único.
- No Hay Orden Definido: A diferencia de List, Set no garantiza un orden específico de los elementos. Los elementos no se almacenan en función de su posición.

- Tamaño Dinámico: Al igual que List, los Set en Java tienen un tamaño dinámico, lo que facilita agregar o eliminar elementos en cualquier momento sin especificar un tamaño fijo.

Implementaciones de Set en Java:

- HashSet: Esta implementación utiliza una tabla hash para almacenar elementos, lo que proporciona un acceso rápido y eficiente para la búsqueda, inserción y eliminación de elementos. No garantiza ningún orden específico.
- LinkedHashSet: Similar a HashSet, pero mantiene un orden de inserción, lo que significa que los elementos se almacenan en el orden en que se agregaron.
- TreeSet: Almacena elementos en un árbol balanceado, lo que garantiza un orden ascendente de los elementos. Es útil cuando se necesita un conjunto ordenado de elementos.

Métodos Importantes en Set:

- add(E elemento): Agrega un elemento al conjunto. Si el elemento ya existe, no se agregará de nuevo.
- remove(Object objeto): Elimina el elemento especificado del conjunto.
- contains(Object objeto): Verifica si el conjunto contiene el objeto dado.
- size(): Devuelve el número de elementos en el conjunto.
- isEmpty(): Verifica si el conjunto está vacío.
- clear(): Elimina todos los elementos del conjunto.

Eficiencia y Uso de Set en Comparación con Otros Tipos de Colecciones:

- Elementos Únicos: La principal ventaja de Set es su capacidad para almacenar elementos únicos. Si necesitas mantener una colección sin duplicados, Set es la elección adecuada.
- Acceso Eficiente: Las implementaciones de Set, como HashSet, ofrecen un acceso eficiente para verificar la existencia de elementos y agregar o eliminar elementos.
- Orden: Si necesitas un conjunto ordenado, puedes optar por LinkedHashSet o TreeSet, dependiendo de si deseas mantener el orden de inserción o un orden ascendente de elementos.
- Escenarios Específicos: La elección entre las implementaciones de Set dependerá de tus necesidades específicas. HashSet es eficiente en la

mayoría de los casos, mientras que LinkedHashSet y TreeSet son útiles en situaciones donde se necesita un cierto orden.

Queue:

En Java, Queue es una interfaz que representa una colección de elementos ordenados en una estructura de cola. Los elementos se insertan al final de la cola y se eliminan del principio de la cola, lo que sigue el principio "FIFO" (Primero en Entrar, Primero en Salir). Aquí se destacan sus características y los métodos más utilizados e importantes para operaciones comunes:

Características Principales de Queue en Java:

- FIFO: Los elementos se manejan siguiendo el principio FIFO, lo que significa que el primer elemento en ser agregado es el primero en ser eliminado.
- No Permite Duplicados: En una cola típica, no se permiten elementos duplicados. Cada elemento debe ser único.
- Tamaño Dinámico: Al igual que List y Set, las colas en Java tienen un tamaño dinámico, lo que facilita agregar o eliminar elementos en cualquier momento sin especificar un tamaño fijo.

Implementaciones de Queue en Java:

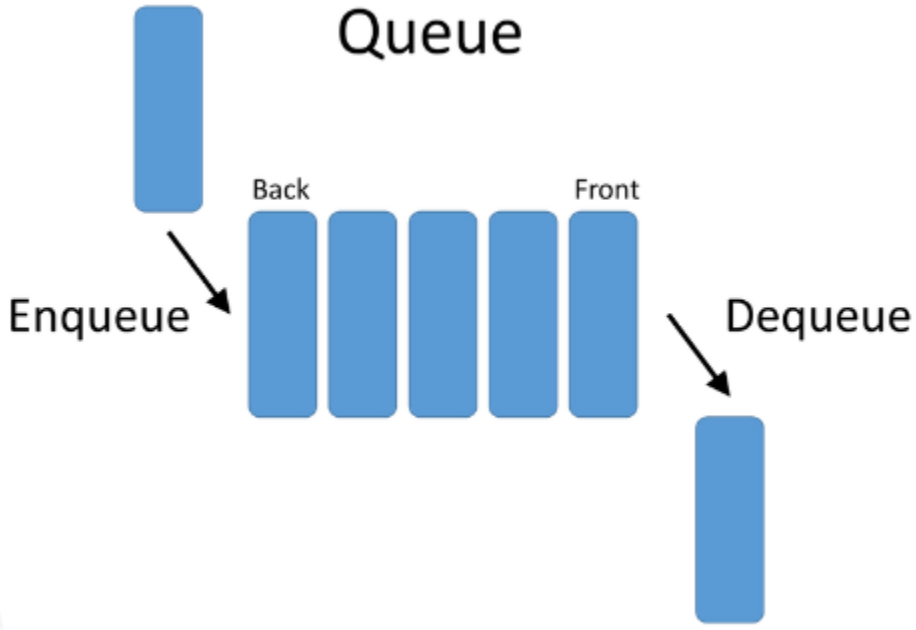
- LinkedList: La clase LinkedList en Java se puede utilizar para implementar una cola (Queue). Proporciona un rendimiento eficiente para la mayoría de las operaciones de cola.
- PriorityQueue: Esta implementación utiliza una cola de prioridad para mantener los elementos ordenados según un criterio específico. Los elementos se eliminan en función de su prioridad.

Métodos Importantes en Queue:

- offer(E elemento): Agrega un elemento al final de la cola. Devuelve true si la operación fue exitosa, false si la cola está llena (en caso de colas limitadas).
- poll(): Elimina y devuelve el elemento en el frente de la cola. Si la cola está vacía, devuelve null.
- peek(): Devuelve el elemento en el frente de la cola sin eliminarlo. Si la cola está vacía, devuelve null.
- size(): Devuelve el número de elementos en la cola.
- isEmpty(): Verifica si la cola está vacía.
- clear(): Elimina todos los elementos de la cola.

Eficiencia y Uso de Queue en Comparación con Otros Tipos de Colecciones:

- FIFO: La principal ventaja de una cola (Queue) es su capacidad para manejar elementos en orden de llegada, lo que es útil en escenarios como administración de tareas y procesos.
- Prioridad: PriorityQueue es útil cuando necesitas mantener elementos ordenados en función de una prioridad específica.
- Implementación: LinkedList es una implementación flexible que es adecuada para la mayoría de las operaciones de cola. PriorityQueue se utiliza cuando se requiere orden por prioridad.
- Escenarios Específicos: La elección entre LinkedList y PriorityQueue dependerá de las necesidades específicas de tu aplicación, considerando el orden y la prioridad de los elementos en la cola.



Map:

En Java, Map es una interfaz que representa una colección de pares clave-valor, donde cada clave está asociada a un valor único. Los mapas son ampliamente utilizados para almacenar datos en forma de pares clave-valor y permiten un acceso eficiente a los valores utilizando las claves correspondientes. Aquí se destacan sus características y los métodos más utilizados e importantes para operaciones comunes:

Características Principales de Map en Java:

- Pares Clave-Valor: Un mapa almacena datos en forma de pares clave-valor, donde cada clave está asociada a un valor. Cada clave es única en el mapa.
- No Permite Claves Duplicadas: No se permiten claves duplicadas en un mapa. Cada clave debe ser única y está asociada a un único valor.
- Tamaño Dinámico: Al igual que List, Set y Queue, los mapas en Java tienen un tamaño dinámico, lo que facilita agregar o eliminar pares clave-valor en cualquier momento sin especificar un tamaño fijo.

Implementaciones de Map en Java:

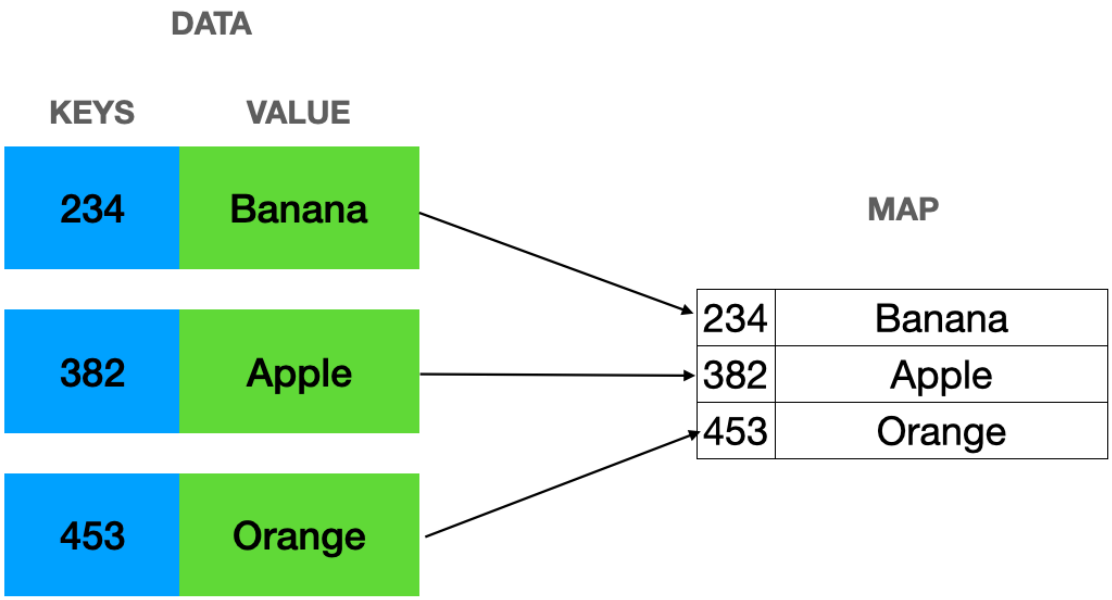
- HashMap: Esta implementación utiliza una tabla hash para almacenar pares clave-valor, lo que proporciona un acceso rápido a los valores a través de las claves. No garantiza ningún orden específico de los elementos.
- LinkedHashMap: Similar a HashMap, pero mantiene un orden de inserción, lo que significa que los elementos se almacenan en el orden en que se agregaron.
- TreeMap: Almacena pares clave-valor en un árbol balanceado, lo que garantiza un orden ascendente de las claves.

Métodos Importantes en Map:

- put(K clave, V valor): Agrega un par clave-valor al mapa o reemplaza el valor si la clave ya existe.
- get(Object clave): Obtiene el valor asociado con la clave especificada. Si la clave no existe, devuelve null.
- remove(Object clave): Elimina el par clave-valor asociado con la clave especificada.
- containsKey(Object clave): Verifica si el mapa contiene una clave específica.
- containsValue(Object valor): Verifica si el mapa contiene un valor específico.
- size(): Devuelve el número de pares clave-valor en el mapa.
- isEmpty(): Verifica si el mapa está vacío.
- clear(): Elimina todos los pares clave-valor del mapa.

Eficiencia y Uso de Map en Comparación con Otros Tipos de Colecciones:

- Pares Clave-Valor: La principal ventaja de un mapa (Map) es su capacidad para asociar valores únicos con claves correspondientes, lo que permite un acceso rápido y eficiente a los valores mediante las claves.
- Implementaciones: La elección de la implementación de mapa depende de tus necesidades. HashMap es eficiente en la mayoría de los casos y proporciona acceso rápido. LinkedHashMap mantiene el orden de inserción, mientras que TreeMap garantiza un orden ascendente de las claves.
- Escenarios Específicos: La elección entre las implementaciones de mapa dependerá de las necesidades específicas de tu aplicación, considerando el orden, el acceso a los valores y otros factores de rendimiento. Si necesitas acceso rápido por clave y no necesitas un orden específico, HashMap es una elección común. Si requieres un cierto orden, puedes optar por LinkedHashMap o TreeMap.



Iterator:

El Iterator es una interfaz en Java que se utiliza para recorrer elementos de colecciones, como List, Set, y Map, de manera secuencial. Proporciona un medio uniforme para acceder a elementos en una colección sin exponer los detalles de implementación subyacentes de la colección. El uso de Iterator es común en el contexto de las colecciones para realizar operaciones de lectura, búsqueda y eliminación de elementos.

Principales Características del Iterator:

1. Iteración Secuencial: El Iterator permite recorrer los elementos de una colección de forma secuencial, uno a uno, comenzando desde el primer elemento y avanzando hasta el último.
2. Métodos Básicos: Los métodos más comunes proporcionados por la interfaz Iterator son hasNext(), next(), y remove().
    - hasNext(): Verifica si hay más elementos en la colección para iterar.
    - next(): Devuelve el siguiente elemento en la colección y avanza el iterador al siguiente elemento.
    - remove(): Permite eliminar el elemento actual de la colección. Este método no está disponible en todos los tipos de iteradores y puede lanzar una excepción si se utiliza incorrectamente.
3. Independiente de la Implementación: Los detalles de cómo se implementa la iteración están ocultos para el usuario, lo que permite cambiar la estructura subyacente de la colección sin afectar el código que utiliza el Iterator.
4. Uso con Colecciones: Se utiliza comúnmente en combinación con colecciones como List, Set, y Map para recorrer, buscar y, en algunos casos, eliminar elementos de la colección.

Relación con las Colecciones:

El Iterator se relaciona estrechamente con las colecciones en Java en los siguientes aspectos:

1. Obtención de Iteradores: Las colecciones proporcionan un método llamado iterator() que devuelve un objeto Iterator que se puede utilizar para recorrer los elementos de la colección. Por ejemplo, puedes obtener un iterador para una lista de la siguiente manera:

```
List<String> lista = new ArrayList<>();

Iterator<String> iterador = lista.iterator();
```

2. Recorrido de Elementos: Utilizando el Iterator, puedes recorrer los elementos de la colección uno por uno usando el método next() hasta que hasNext() indique que no hay más elementos.

```
while (iterador.hasNext()) {

    String elemento = iterador.next();

     // Realiza alguna operación con el elemento
```

```
}
```

3. Eliminación de Elementos: En algunos tipos de colecciones, como List, puedes utilizar el método remove() del Iterator para eliminar elementos de la colección de manera segura durante la iteración.

```
while (iterador.hasNext()) {
    String elemento = iterador.next();
    if (condición) {
            iterador.remove(); // Elimina el elemento actual de la colección
    }
}
```